Modern Engineering

Day 1

Prudential Financial

GENERAL ASSEMBLY

Whilst we wait....

In chat write:

Your name, role, and a recent impulse buy.

Meet General Assembly

Your Instructor

Ben Piper

Author and Consultant, Coastal Carolina Networks

- Ben Piper is the author of several Sybex study guides, including the *AWS Certified Solutions Architect Study Guide*, *AWS Certified Cloud Practitioner Study Guide*, *CompTIA Cloud+ Study Guide,* and *CCNP Enterprise Certification Study Guide*. He has also created 40+ Pluralsight courses covering AWS, networking, and configuration management, and wrote *Learn Cisco Network Administration in a Month of Lunches* [Manning].
- Prior to this, he has had a diverse career spanning retail, finance, and healthcare industries, where he performed network, Linux, Windows, Citrix, and VMware administration.

Meet General Assembly

Your Instructional Associate

Troy Swayzee

Software Engineer

- Troy has supported Consumer learners as Lead Instructor for General Assembly's Software Engineering Immersive programs since 2021.
- He spent over 3 years as a Software Engineer at Archbright engineering BI/IT integrations, applications, infrastructure and tools that enhanced the company's operations and services.

Meet General Assembly

Your Instructional Associate

Mario Recinos

GA Instructor and Software Engineer

- Mario is a dedicated Software Engineer and passionate educator, committed to empowering underserved communities with technology resources.
- He is a proud graduate of General Assembly SEI, 2021 before joining General Assembly Instructional Associate for six cohorts over the past two years.
- Previously, Mario garnered valuable professional experiences as a Software Engineer and Ambassador for Y-Combinator startup. He mentored individuals from non-traditional backgrounds guiding them to break into the tech industry to actively strive to bridge the technology divide for underserved communities.

Remember Your To Do List

- Be engaged and ask questions
  Start every session with your microphone **muted** and **camera on**
- Communicate with us
- Don’t be afraid of breaking things
- Don’t worry about writing things down
- Try not to get distracted by your inbox or phone calls
- Help your peers out

You will receive digital copies of the slides in your inbox after today.

Our Goal

The goal of this seminar is to introduce approaches, architectures, and patterns to add to your toolbox.

It is up to you to decide **HOW** to use these tools.

There are no absolute rights and wrongs.

Need to Know?

You don’t need to know and understand every legacy or modern IT system, application, network protocol, database technology, or programming language. Even experts in a very specific field don’t know everything about every aspect of their specialism.

You do need to grasp the core concepts and terminology and be able to identify some of the hallmarks of legacy and more modern architectures.

Any Questions

PROGRAM OVERVIEW | Modern Engineering Fundamentals

Instructor-Led | Live Online | ~ 160 Hours | ~ 4 Weeks

Overview:

The purpose of this course is to prepare participants to be comfortable working in more modern architecture paradigms, **whilst developing the skills to enable them build a production ready secure API using modern tools and practices**

Prior To Course

3 Weeks Prior: Assessments and Pre Learning
1 Week Prior: 1 hour virtual installfest and pre-install instructional doc

Kickoff

MODULE 1 : Architecture & Requirements

2 Days

Legacy Systems & The Cloud
Cloud Architectural Models
Microservices
Web Application Security
Encryption at Rest
Encrypted Active Data
Authorization, Roles Permissions
Configuration & Secrets
Exploring the Model Application API with Postman

MODULE 2 : Building the API

3-4 Days

One Datastore per Microservice
Relational vs. Non-Relational
Choosing Your Database
Schema Management
Saga
Introducing Axios: JavaScript Request Library
Build APIs From Scratch

MODULE 3 : Building the Front End

3-4 Days

Introduction to React
React Components
React Props
React Hooks
React Styling Approaches
React Routing
Axios
Build missing Model Application UI

MODULE 4 : Test-Driven Development

3-5 Days

Test Driven Development
Beyond TDD
Mocks, Stubs, Fakes
Introduction to Jest
Introduction to Selenium
Testing a Function and API's
Design Unit Tests
Build the Unit Tests

MODULE 5 : CI/CD, Docker & AWS

3-5 Days

Review React App
Docker & Containerization
CI/CD
CI/CD with Jenkins – Demo
Introduction to Cloud Formation
Criticality of GOOD Logging

3-4 Days

Case Study
***Capstone Development**
***Capstone Presentations**

Review

Week 1 Class Schedule: APIs and Databases in JavaScript

	Day 1	Day 2	Day 3	Day 4	Day 5
9:00 AM	Legacy Systems & the Cloud	Testing APIs with Postman	SQL	APIs with Express To-Do List	JSON Web Tokens
10:00 AM	Web Application Security	Postman Application	Build Raw SQL Queries		Lab - Complete Gold Codes
11:00 AM					
12:30 PM			**Lunch**		
1:30 PM	Microservices	SQL Intro	Lab - SQL queries with Carmen Sandiego	Lab - Solo API	Related Models
2:00 PM	Full Environment Set-Up	Superstore Exploration			Users + ToDos
3:00 PM					
4:00 PM					
5:00 PM	Review	Review	Review	Review	Review

Lesson - Instruction	Instructor-Led Lecture, Guided Walk Through, and Modelling
Practice - Application	Solo Work Pair Activity Group Activity

Week 2 Class Schedule: React, Intro to TDD

	Day 6	Day 7	Day 8	Day 9	Day 10
9:00 AM	Introducing React	React State and Hooks	React Unidirectional Data Flow	Full Stack React	TDD in JavaScript
10:00 AM					
11:00 AM	React Hello World	Lab - Weather API	Fruits Filter Code Along	Lab - ToDos App	Lab - Unit Tests with Jest
12:30 PM	**Lunch**				
1:30 PM	Components and Props	Styling in React	React Router	Solo Full-Stack App	API Testing with SuperTest
2:00 PM	Lab - Passing Data Around	Lab - Add Material UI to components	Routing Lab		Lab - API Tests
3:00 PM					
4:00 PM					
5:00 PM	Review	Review	Review	Review	Review

Lesson - Instruction	Instructor-Led Lecture, Guided Walk Through, and Modelling
Practice - Application	Solo Work Pair Activity Group Activity

Week 2 Class Schedule: React, Intro to TDD

	Day 6	Day 7	Day 8	Day 9	Day 10
9:00 AM	Introducing React	React State and Hooks	Styling in React	React Router	Solo Full-Stack App
10:00 AM					
11:00 AM	React Hello World	Lab - Weather API	Lab - Add Material UI to components	Routing Lab	Solo Full-Stack App Continued
12:30 PM	**Lunch**				
1:30 PM	Components and Props	Intro to Fetch	React Unidirectional Data Flow	Full Stack React	TDD in Javascript
2:00 PM	Lab - Passing Data Around	Lab - Weather API	Fruits Filter Code Along	Lab - ToDos App	Lab - Unit Tests with Jest
3:00 PM					
4:00 PM					
5:00 PM	Review	Review	Review	Review	Review

Lesson - Instruction	Instructor-Led Lecture, Guided Walk Through, and Modelling
Practice - Application	Solo Work Pair Activity Group Activity

Week 3 Class Schedule: TDD, Containers

	Day 11	Day 12	Day 13	Day 14	Day 15
9:00 AM	Load Testing, Mocks, Stubs, Fakes	Selenium Testing	Docker and Containerization	Jenkins Ci/CD Pipelines	AWS Cloud Formation Template Guest Demo
10:00 AM	Pair Programming TDD Escape Room	Lab - Selenium Tests	Lab - Containerize ToDo List API	Lab - ToDo Pipeline	
11:00 AM					
12:30 PM	**Lunch**				
1:30 PM	UI Testing with Jest	Create Selenium and Unit Tests for a previous App	Dockerize an Application	AWS DynamoDB NoSql vs SQL research discussion	Deployment to Prudential AWS Guest Demo
2:00 PM	Lab - Tests with React and Jest			Lab - Replace SQL with Dynamo in an API	
3:00 PM					
4:00 PM					
5:00 PM	Review	Review	Review	Review	Review

Lesson - Instruction	Instructor-Led Lecture, Guided Walk Through, and Modelling
Practice - Application	Solo Work Pair Activity Group Activity

Week 3 Class Schedule: TDD, Containers

	Day 11	Day 12	Day 13	Day 14	Day 15
9:00 AM	API Testing With Supertest	UI Testing with Jest	Docker and Containerization	Jenkins Ci/CD Pipelines	AWS Cloud Formation Template Guest Demo
10:00 AM					
11:00 AM	Lab - API Testing	Lab - Tests with React and Jest	Lab - Containerize ToDo List API	Lab - ToDo Pipeline	
12:30 PM	**Lunch**				
1:30 PM	Load Testing, Mocks, Stubs, Fakes	Selenium Testing	Dockerize an Application	AWS DynamoDB NoSql vs SQL research discussion	Deployment to Prudential AWS Guest Demo
2:00 PM					
3:00 PM	Pair Programming TDD Escape Room	Lab - Selenium Tests + Create Selenium and Unit Tests for a previous App		Lab - Replace SQL with Dynamo in an API	
4:00 PM					
5:00 PM	Review	Review	Review	Review	Review

Lesson - Instruction	Instructor-Led Lecture, Guided Walk Through, and Modelling
Practice - Application	Solo Work Pair Activity Group Activity

Week 4 Class Schedule: Case Study and Capstone

	Day 16	Day 17	Day 18	Day 19	Day 20
9:00 AM	Full Review Code-Along	Capstone Project	Capstone Project	Capstone Project	Capstone Project
10:00 AM					
11:00 AM					
12:30 PM	Lunch				
1:30 PM	Full Review Code-Along	Capstone Project	Capstone Project	Capstone Project	Capstone Review
2:00 PM					
3:00 PM					
4:00 PM					
5:00 PM	Review	Review	Review	Review	Review

Lesson - Instruction	Instructor-Led Lecture, Guided Walk Through, and Modelling
Practice - Application	Solo Work Pair Activity Group Activity

LESSON ROADMAP

Discover Cloud Computing

Moving to the Cloud

Traditional to Modern Applications

Microservices

Virtual Machine Set-Up

Bring It Home

MEF MODULE 1 DAY 1: Architecture & Requirements

Schedule	
9:00–9:15 am	Welcome and Warm-Up
9:15–10:30 am	**Legacy Systems and the Cloud**
10:30 am–11:15 pm	**Web Application Security**
11:15–12:30 pm	**The Model Application**
12:30–1:30 pm	Lunch
1:30-3:00 pm	**Environment Set Up**
3:00–4:00 pm	**Code Exploration of the DMV App**
4:00 - 4:50 pm	API Design Considerations
4:50–5:00 pm	Bring It Home

LEARNING OBJECTIVES

1. Articulate the benefits and drawbacks of moving to the cloud.
2. Identify major concepts involved with cloud computing.
3. Leverage the best concepts and practices for modern cloud applications.
4. Remove doubts, fears, and reservations associated with cloud computing.

Modern Engineering

Legacy Systems & The Cloud

Hallmarks of Legacy Architecture

Application Silos

➔ Apps that were never designed to interoperate remain inflexible “black boxes”

Monolithic Architecture

➔ Can lead to slow release cycles and more complex deployments

Decentralised Data

➔ Inaccessible between departments

➔ Could be stored in anything from excel spreadsheets to untamed databases

Inefficient Data Processes

➔ Including time consuming backups to physical media on site or off

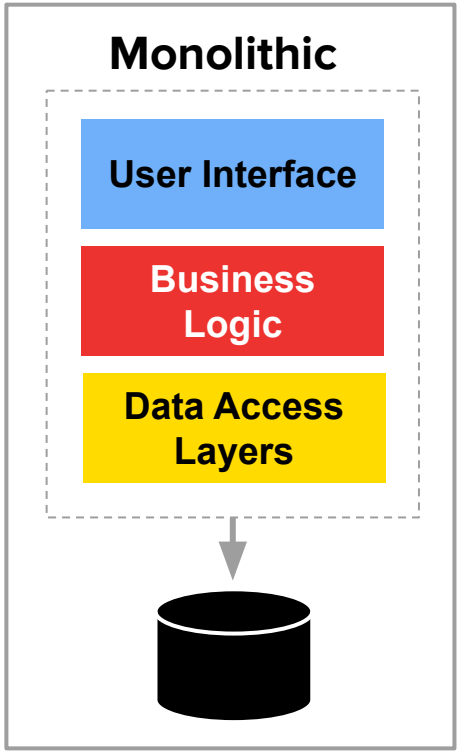


Image 1: Hallmarks of Monolithic Architecture Diagram

Benefits of Legacy Systems

Durable.

- A legacy system can be notoriously durable, supporting essential business processes throughout a lifetime.
- This makes them virtually indispensable for many companies.

Gives customers what they're used to.

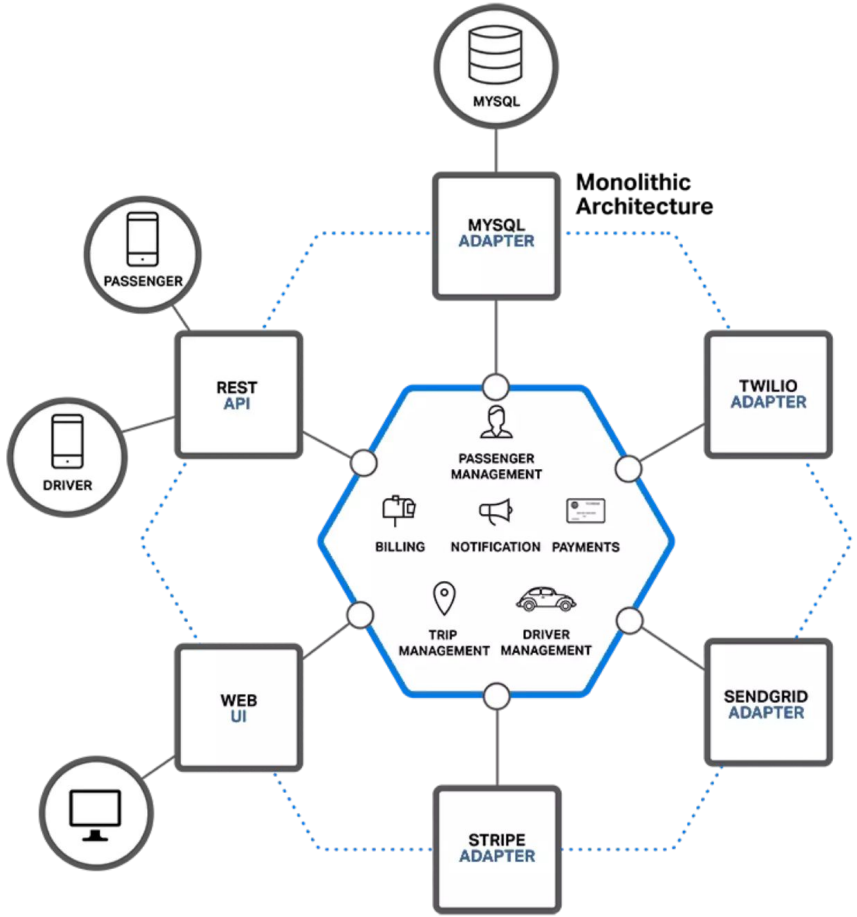
- This may be a primary concern for many service-based enterprises.
- Giving customers a consistent experience, without major changes over the years, is a big advantage.

Modern Engineering

Cloud Architectural Models

The Monolith

The application is a single self-contained unit and does not rely much on other applications.



Source: Viron IT

Image 2: Monolithic Architecture Diagram

The Good

- A single codebase to manage
- Easier to cross-share information
- Centralized control of all data
- Simpler deployment
- Easier end-to-end testing
- Performance

The Bad

- Harder to scale
- Single point of failure
- Change has more risk
- Slow development speed at scale
- Reliability
- Lack of flexibility
- Higher risk to deployment

Monoliths are NOT a bad thing!

There are many circumstances where they are more reliable and easier to maintain than other patterns.

The Service-Oriented Architecture

- A transitional design between a classic monolith and micro-services.
- Actions are broken out into discrete services.
- A common database may be used and one deployment may house multiple services.

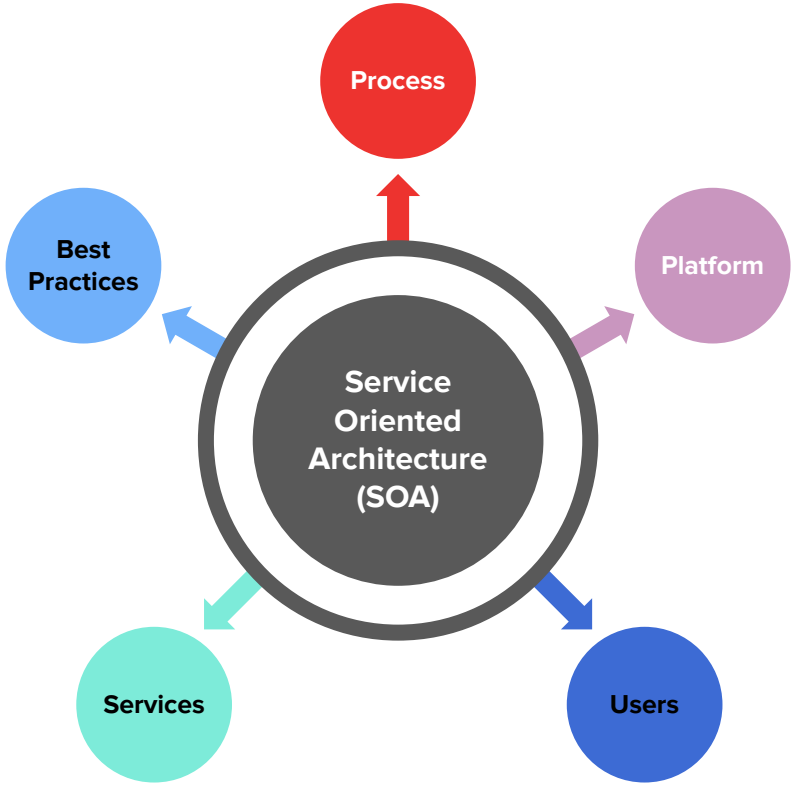


Image 3: The Service-Oriented Architecture Diagram

Service-Oriented Architecture

The Good

- Somewhat simpler codebase to manage
- Stronger separation between display and logic
- Easier to replace components in the future
- Easier to cross-share information
- Centralized control of all data
- Performance

The Bad

- Services can still be complex
- Common data stores create cross-dependencies
- Will eventually evolve into a monolith-like codebase

SOME MAJOR BENEFITS

Reliability

Cloud services are built from the ground up to be fault tolerant and are spread across multiple physical locations.

Speed

Computing resources can be provisioned in minutes with just a few clicks.

Productivity

Teams are able to bypass hardware setup, software patching and other time-consuming IT chores to focus on delivering value and solving business problems.

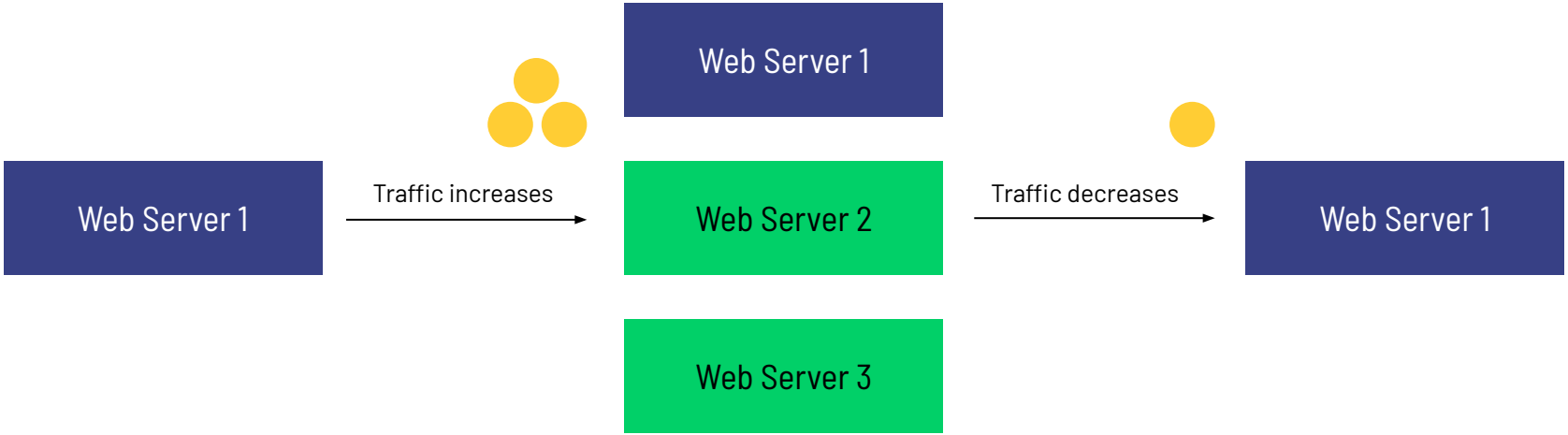
Security

Automatically leverage the expertise and resources of large tech organizations in regards to securing infrastructure.

Cost Savings

Cloud computing eliminates the capital expense of buying hardware / software and maintaining it.

SCALABILITY

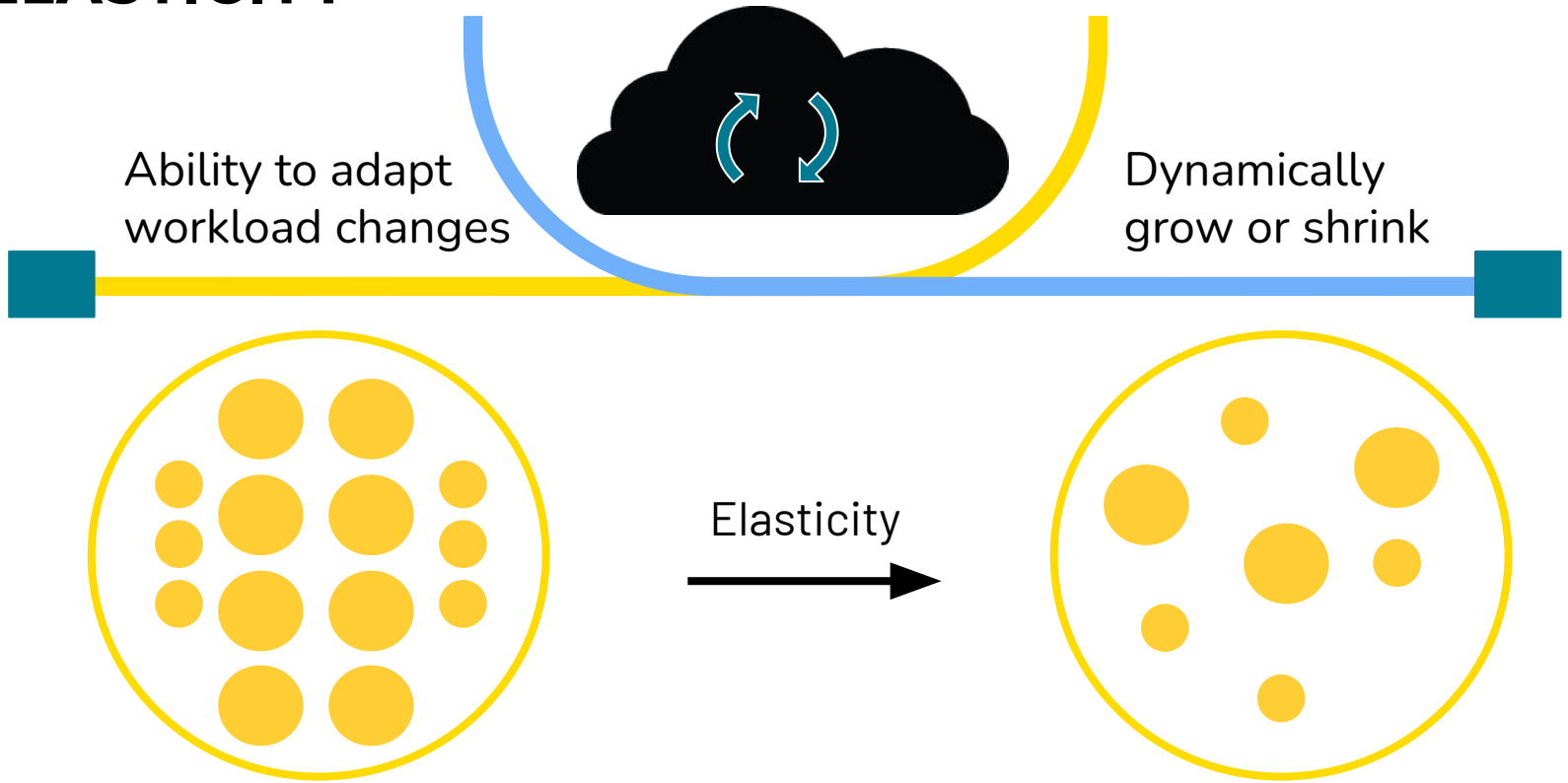


Web servers 2 and 3 are automatically added as needed and removed when they're no longer necessary to save on costs.

With a capex model (owning the hardware onsite) it was a question of putting your finger in the air to anticipate shifting storage requirements. Now we can see what we need immediately and accommodate changes very easily in the most cost-effective manner.

Paul Brown, IT Director for Infrastructure & Support, Cordant Group

ELASTICITY



3 Min

Elasticity: Who Needs It?

What kinds of business might benefit from elasticity?

Come off mute or raise your hand to share your answers.

KNOWLEDGE CHECK

By switching to the AWS Cloud, our uptimes went from around 95 percent to 99.999 percent. We no longer run the risk of downtime impacting revenues.

Jinwei Lin, Assistant Manager, SETTour

THE THREE MODELS

Software as a Service (SaaS)

Infrastructure as a Service (IaaS)

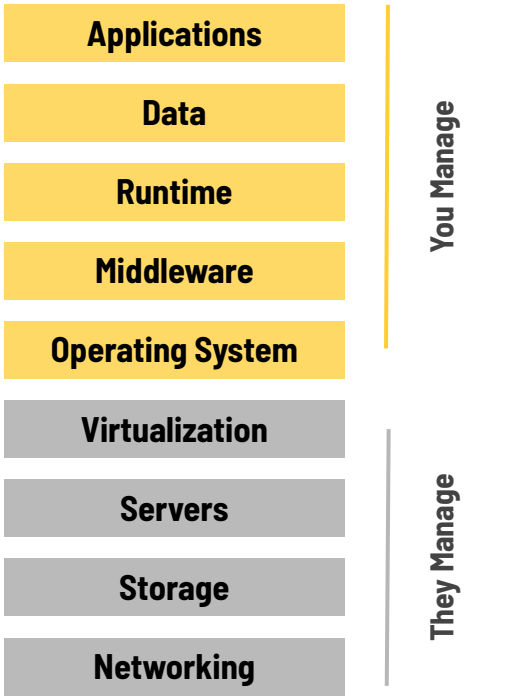
Platform as a Service (PaaS)

INFRASTRUCTURE AS A SERVICE

IaaS provides the same technologies and capabilities as a traditional data center without having to physically maintain or manage all of it.

Examples include:

- Amazon Web Services (AWS)
- Digital Ocean
- Google Compute Engine (GCE)
- Microsoft Azure
- Linode

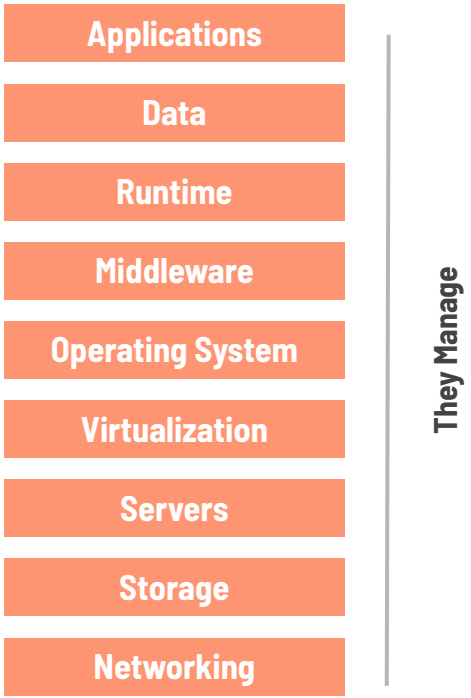


SOFTWARE AS A SERVICE

SaaS is a cloud-based service where instead of downloading software onto your computer to run, you instead access an application via an internet browser.

Examples include:

- Gmail
- Dropbox
- Salesforce
- Confluence

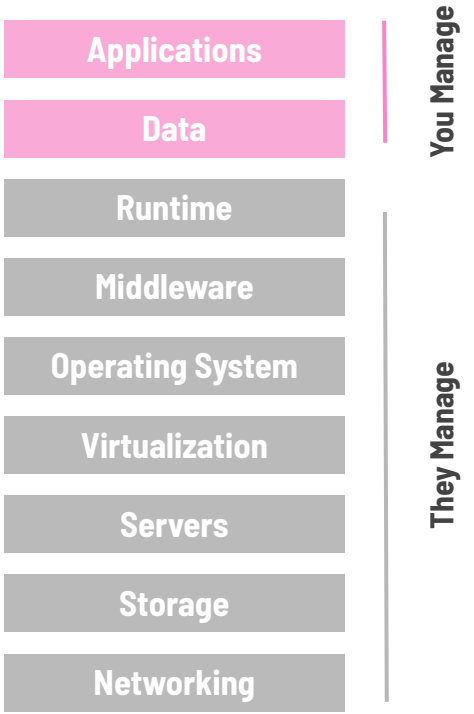


PLATFORM AS A SERVICE

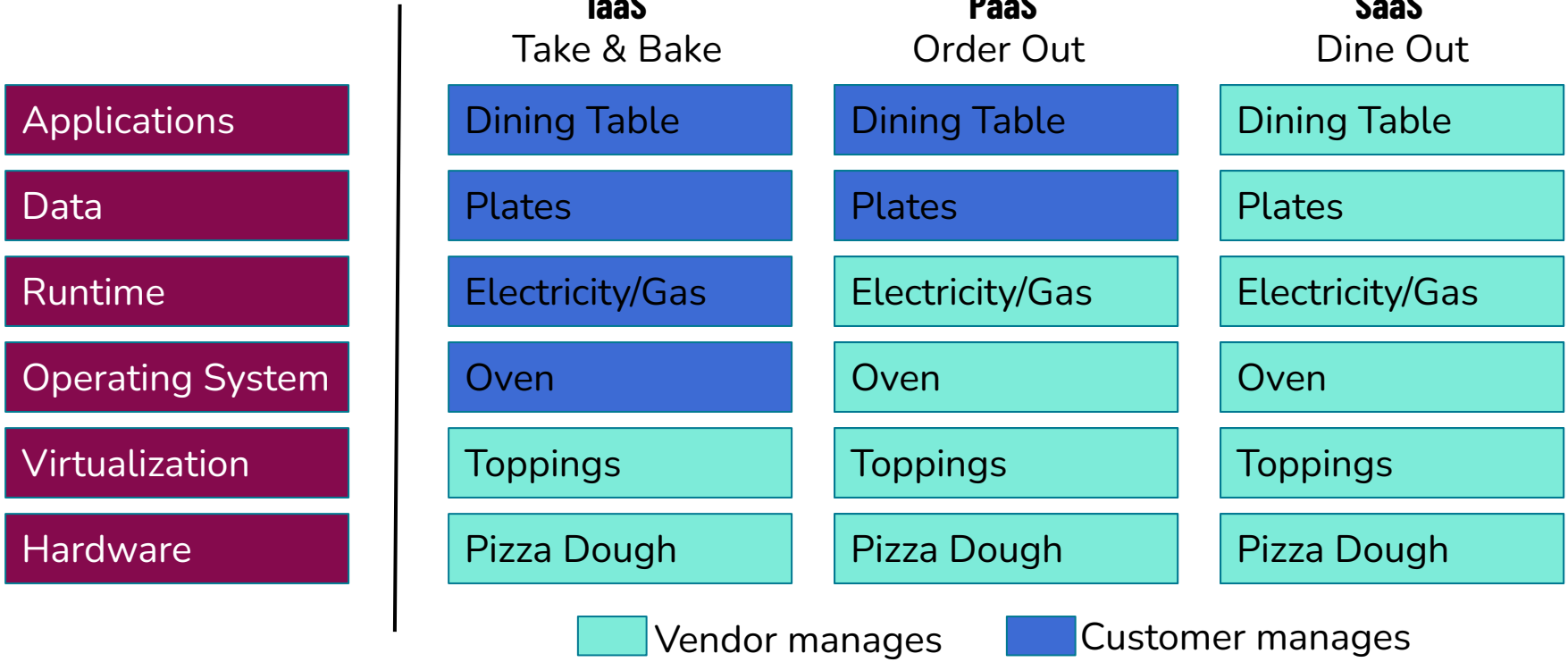
PaaS allows developers the freedom to concentrate on building software without having to worry about operating systems, software updates, storage, or infrastructure.

Examples include:

- AWS Elastic Beanstalk
- Google App Engine
- Heroku



IT'S A SERVICE



DEPLOYMENT MODELS | PUBLIC CLOUD

Public clouds are the most common way of deploying cloud computing.

All the hardware, software, and other supporting infrastructure is owned and managed by a cloud provider such as Microsoft Azure or AWS.

In a public cloud, **you share the same hardware, storage, and network devices with other organizations**.

DEPLOYMENT MODELS | PRIVATE CLOUD

A private cloud consists of computing resources used exclusively by one business or organization.

In a private cloud, the services and infrastructure are maintained on a private network and **the hardware is dedicated solely to your organization**.

In this way, a private cloud can make it easier for an organization to customize its resources to meet specific IT requirements.

DISCUSSIONS: WRAP-UP

5 Min

Wrap-up

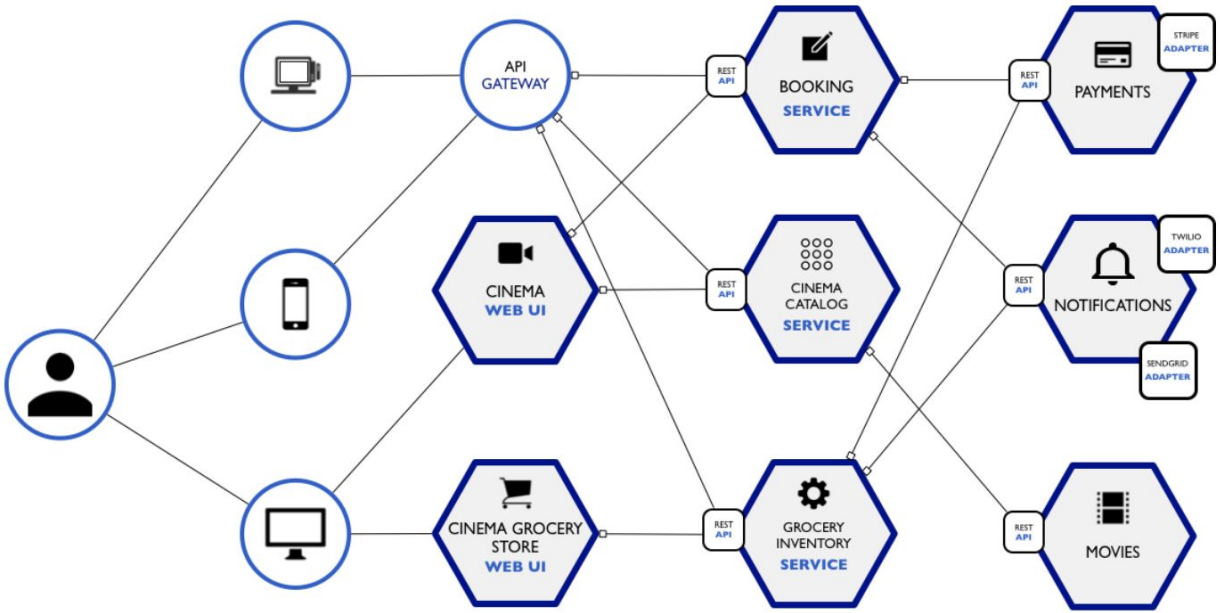
- What is the cloud?
- How does it work?
- What are the different service models?

Come off mute or raise your hand to share your answers.

Modern Engineering

Microservices

The Microservice Architecture



Cinema Microservice Example

Image 4: Cinema Microservice Diagram

- Each service is **independently** deployable.
- Each service has its own logic and database with a **narrow goal**.
- Microservices are intended to manage a **single concern** or function.

Source: medium.com

Microservice Architecture

The Good

- Complexity is made visible
- Services can be separately upgraded and deployed
- Easier to scale development across teams
- Highly scalable infrastructure
- Discrete testing and QA
- Language Independence

The Bad

- Cross-service data queries are slow
- Development environments are complex
- Debugging is more complex across services
- Standardization and clear ownership is MUCH harder
- Increased baseline infrastructure and environment costs

Discussion - Eye on the DMV

Our Model DMV Application was split from a Monolith into:

- 3 backend services each with a separate database (driver, plate, license)
- 1 front end React application service
- 1 HTTP API layer service

What are some advantages and disadvantages to this approach?

Create a Separate Datastore for Each Microservice

Breaking apart the data can make data management more complicated because the separate storage systems can easily get out sync or become inconsistent, and foreign keys can change unexpectedly.

You need to add a tool that performs master data management (MDM), operating in the background to find and fix inconsistencies.

Master Data Management Tools

MDM tools might examine every database that stores subscriber IDs to verify that the same IDs exist in all of them (there aren't missing or extra IDs in any one database). You can write your own tool or buy one.

Many commercial relational database management systems (RDBMSs) do these kinds of checks, but they usually impose **too many requirements for coupling**, and so don't scale.

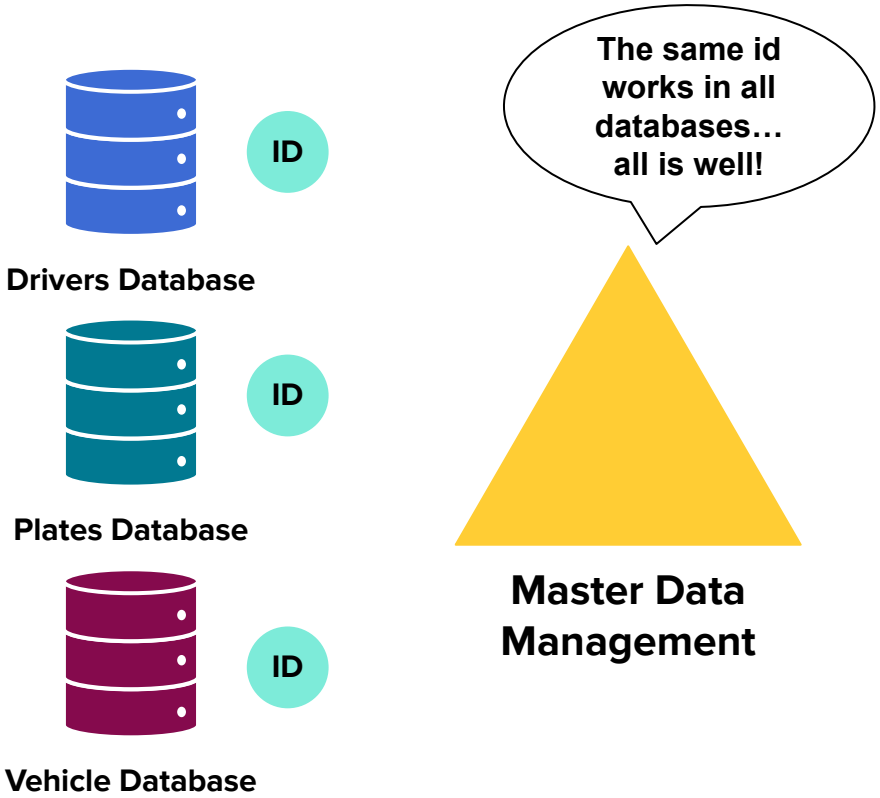


Image 5: Microservice Database Example

Discussion:

Splitting the Database

10 min

One of the biggest differences between classical service-oriented architecture and microservice architecture is that each microservice has a discrete data store. Consider one of the applications you have either built or worked on.

Q: What are the advantages and challenges of splitting the database into discrete databases?

Come off of mute to share or type your answer in the chat.

Do a Separate Build for Each Microservice

Do a separate build for each microservice, so that it can pull in component files from the repository at the revision levels appropriate to it.

Decommissioning old file versions becomes harder and must be very deliberate.

The asymmetry is intentional; you want introducing a new microservice, file, or function to be easy, not dangerous.

Microservices may pull in a similar set of files, but at different revision levels.

Recap

- Introduction to various Architecture patterns
  - Service-Oriented Architecture
  - Monolith
  - Microservices
- Considering Databases for Microservices

Any Questions

Break Time

Up Next

- Security introductions
- Our Model Application DMV introduction/setup

Any Questions

Modern Engineering

Web Application Security

Open Web Application Security Project | Top 10 Security Risks

1. Broken Access Control
2. Cryptographic Failure
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Using Components With Known Vulnerabilities
7. Broken Authentication
8. Software and Data Integrity Failures
9. Insufficient Logging and Monitoring
10. Server Side Request Forgery

Group Exercise:

Research a Security Risk

15 minutes

In small groups of 2-3, research one of the listed top 10 security risks and present to the rest of the class.

Describe what the problem is and, if possible, provide an example of how that problem might be introduced into an application and/or solved.

Security is part of the *architecture*
not something added afterwards

Modern Engineering

Encryption

Encryption at Rest

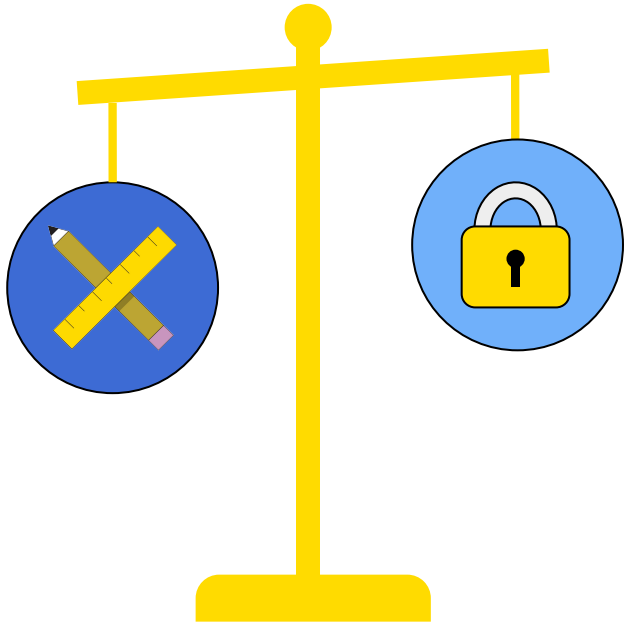
This expansive category includes keeping all drives and files encrypted when not in use.

- BitLocker & FileVault 2 on laptops
- Encryption in cloud volumes
- Encrypted compressed files
- Encrypted backups

Encrypting Data In Use

Keeping data encrypted while in use is an art of balancing security and utility.

- Field-level encryption in the database
- Keeping data encrypted in memory
- Encryption in search indexes



JSON Web Tokens: Passable, Encrypted Data

In the context of our microservices, we'll be using the gold standard of modern authentication protocols: JSON Web Tokens.

JWTs allow our API to confirm requests coming in are authenticated in a way that won't compromise the data if the request is intercepted.

Modern Engineering

Authorization, Roles & Permissions

Who's Allowed To Do What

We need to store information about ***who*** is allowed to do ***what*** in our database.

We also know that regardless of whether we end up with tens, hundreds, or thousands of users, we want to describe who can do what in a way that is expressive but not too complicated.

A Little History

Historically, user authentication and authorization was handled by each individual application.

This means applications would need to:

- Track users and passwords.
- Contain a mechanism to match each user to the functions they were allowed to perform.
- Handle the security of the user/permissions framework.

Let’s look at a few permission types...

Attribute-based Permissions

User-Level Permissions

Role-Based Permissions

User-Level Permissions

User permissions are based on attributes:

- User
- Item

What permissions does this user have?

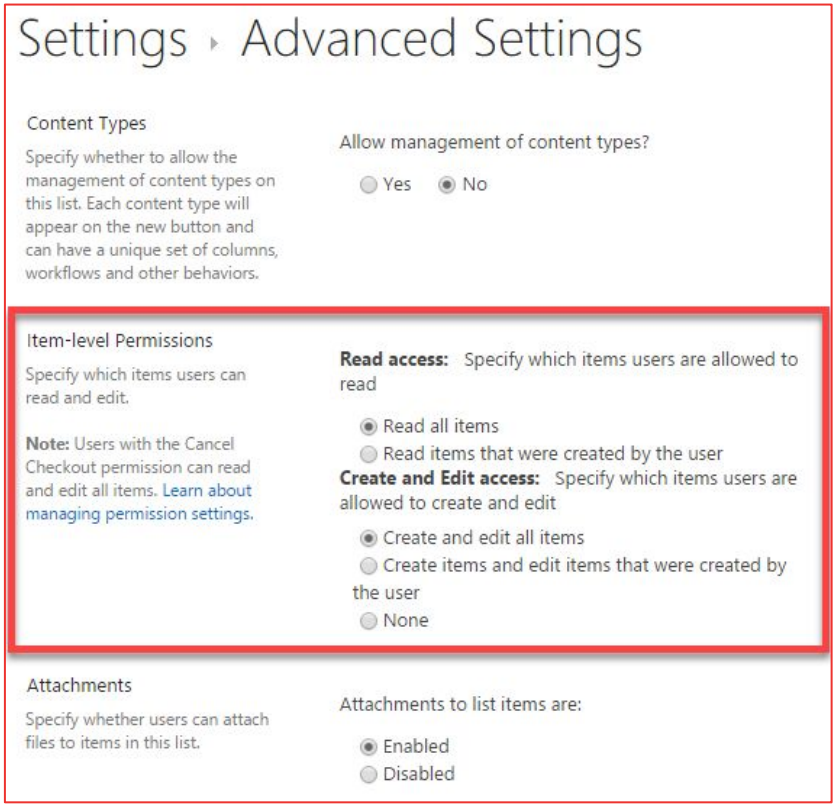


Image 6: User Permissions GUI

Role-Based Permissions

Commonly referred to as "role-based access control" (RBAC).

What benefit does RBAC offer?

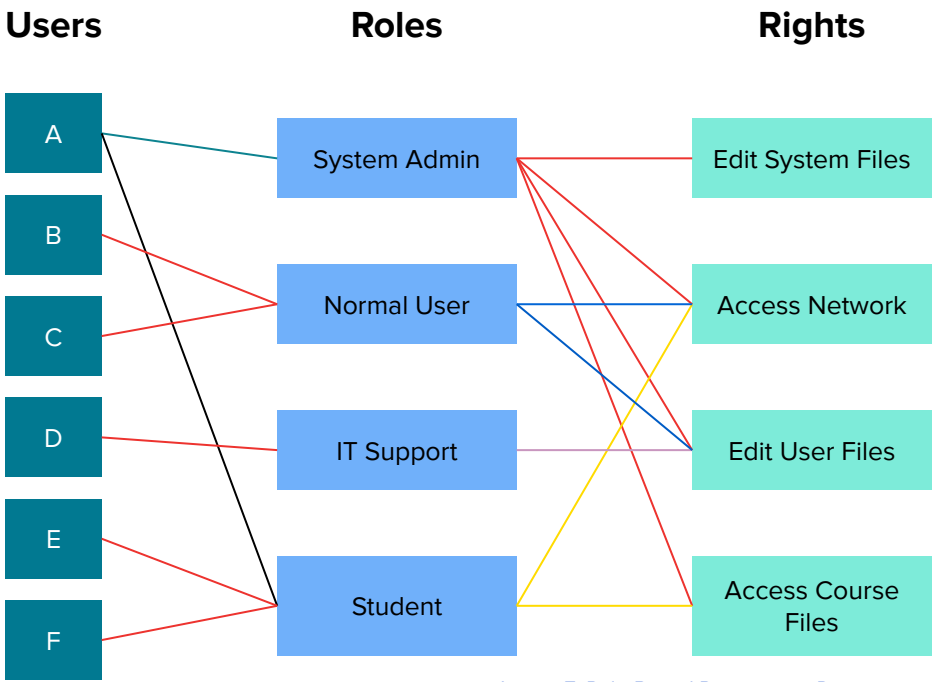


Image 7: Role-Based Permissions Diagram

Attribute-Based Permissions

Attribute based controls are computationally expensive but VERY fine-grained.

A classic example is users being able to edit or delete their own posts on social media, but not those of other users.

What benefit does ABAC offer?

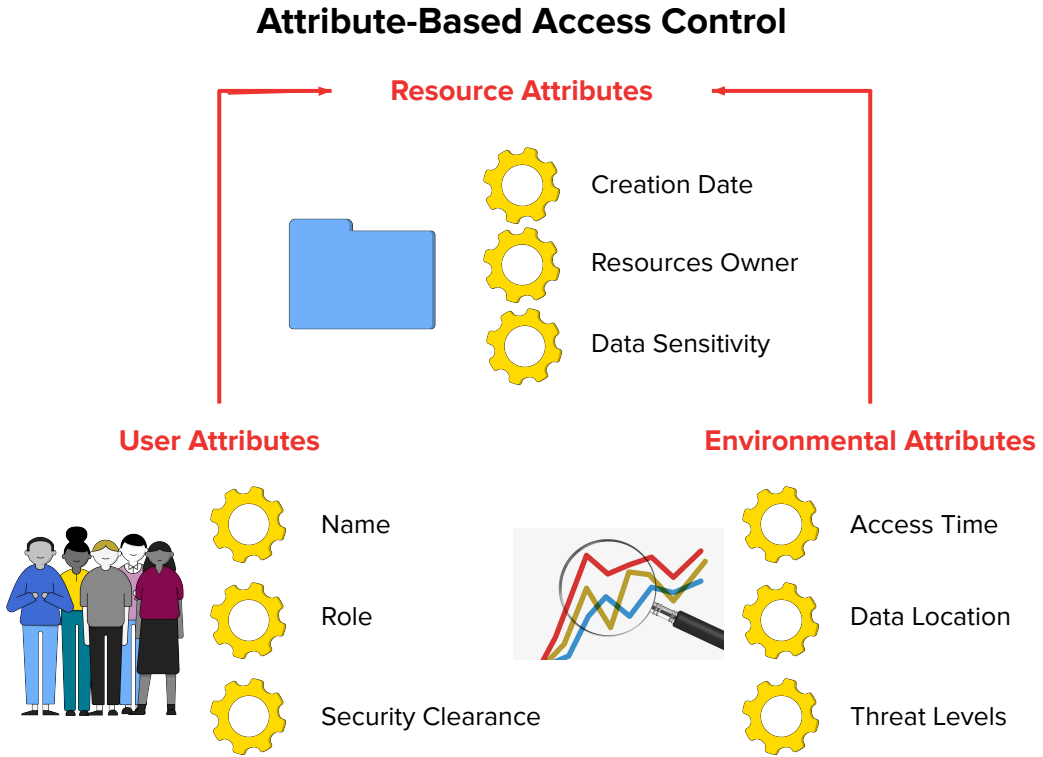


Image 8: Attribute-Based Access Control Flow Chart

Group Exercise:

Authorization, Roles, Permissions

15 minutes

In breakout room groups, discuss which authorizations, roles, and permissions you would apply to the app assigned to your group. We'll come back and compare answers together.

E-commerce Platform:
You are developing an e-commerce platform where users can browse products, add items to their cart, and make purchases. What are the essential roles and permissions needed to ensure secure transactions, protect user data, and manage the overall system?

Financial Management App:
If you're building a financial management application to help users track their expenses and budgets, how would you implement roles and permissions to safeguard sensitive financial data and allow users to collaborate securely with family members or financial advisors?

Travel Booking Platform:
You're creating an online travel booking platform where users can search for flights, hotels, and rental cars. What role-based access controls would you employ to safeguard personal payment information and ensure that only legitimate booking modifications are made?

Modern Engineering

Configuration & Secrets

Configuration Management

DO

- Separate sensitive and non-sensitive values
- Choose a way to retrieve secret values
- Plan how to deploy in different environments
- Plan how to disseminate configurations to different team members and environments

DON'T

- Store sensitive values in git. **EVER!**
- Skip configuration planning to 'just get building'
- Choose a method that doesn't scale well across a cloud environment

Secrets Managers

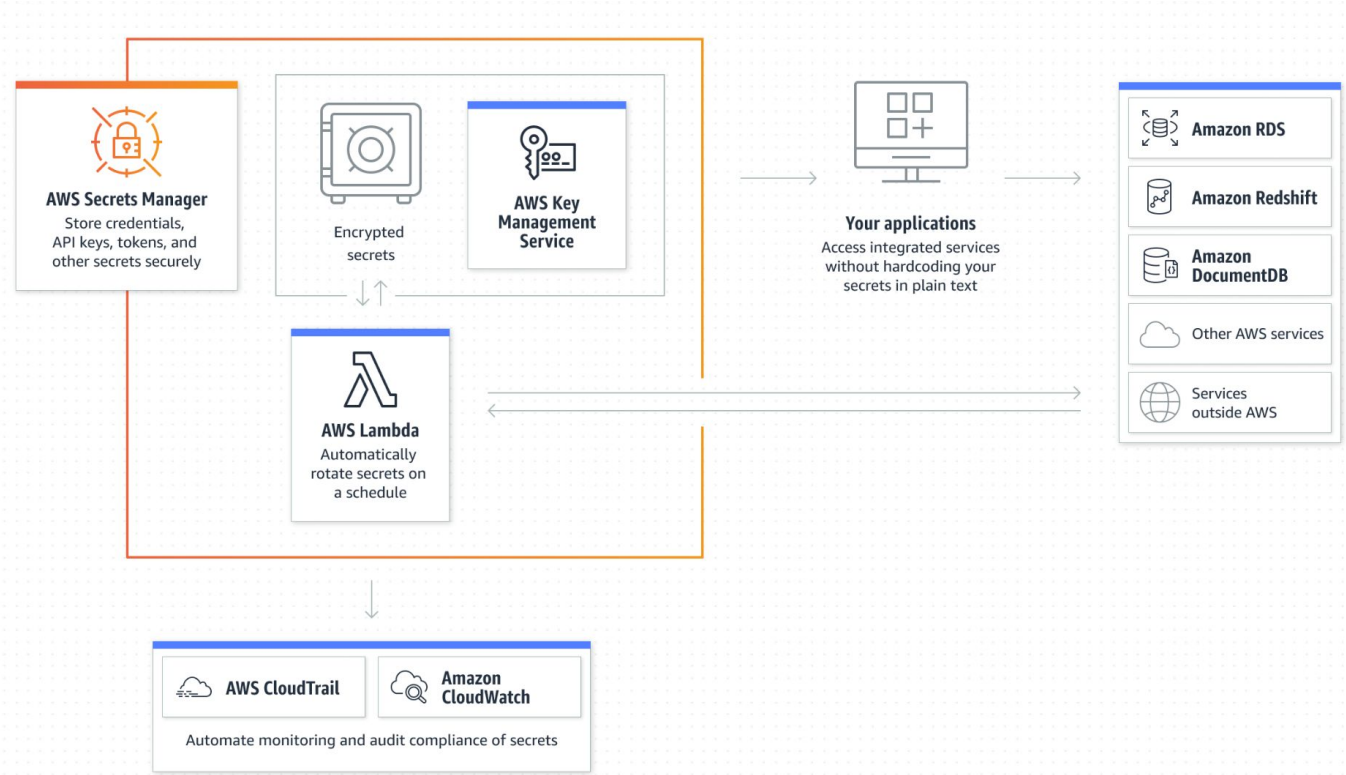


Image 9: Secret Managers Diagram

Modern Engineering

The Model Application

Vehicle Manager

The **Vehicle Manager model** application is a miniature of a system to manage vehicles, drivers, and license plates.

Think of this as a small scale Department of Motor Vehicles (DMV).

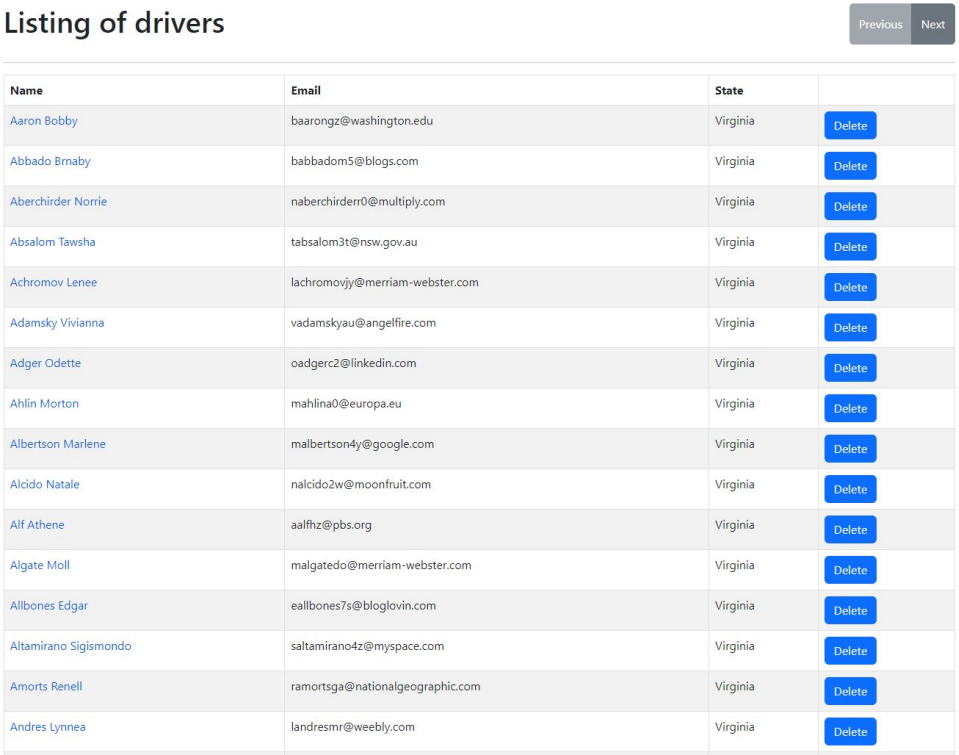


Image 10: Vehicle Managers Example GUI

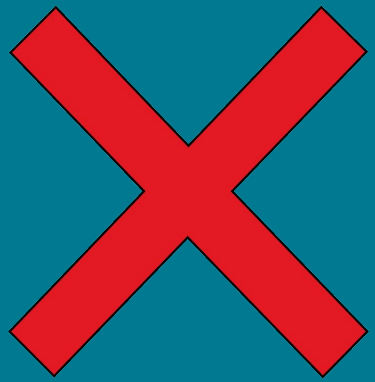
In This Model Application...

- Microservice-based architecture w/3 fully independent services
- Rest interfaces
- Asynchronous patterns
- Saga-pattern data updates
- Independent stores
- Configuration management
- React UI

Not In The Application

(for reasons of time & sanity)

- Authentication & Authorization
- Secrets Management
- Step Functions & Orchestration
- GraphQL



Application Architecture: What We Are Simulating

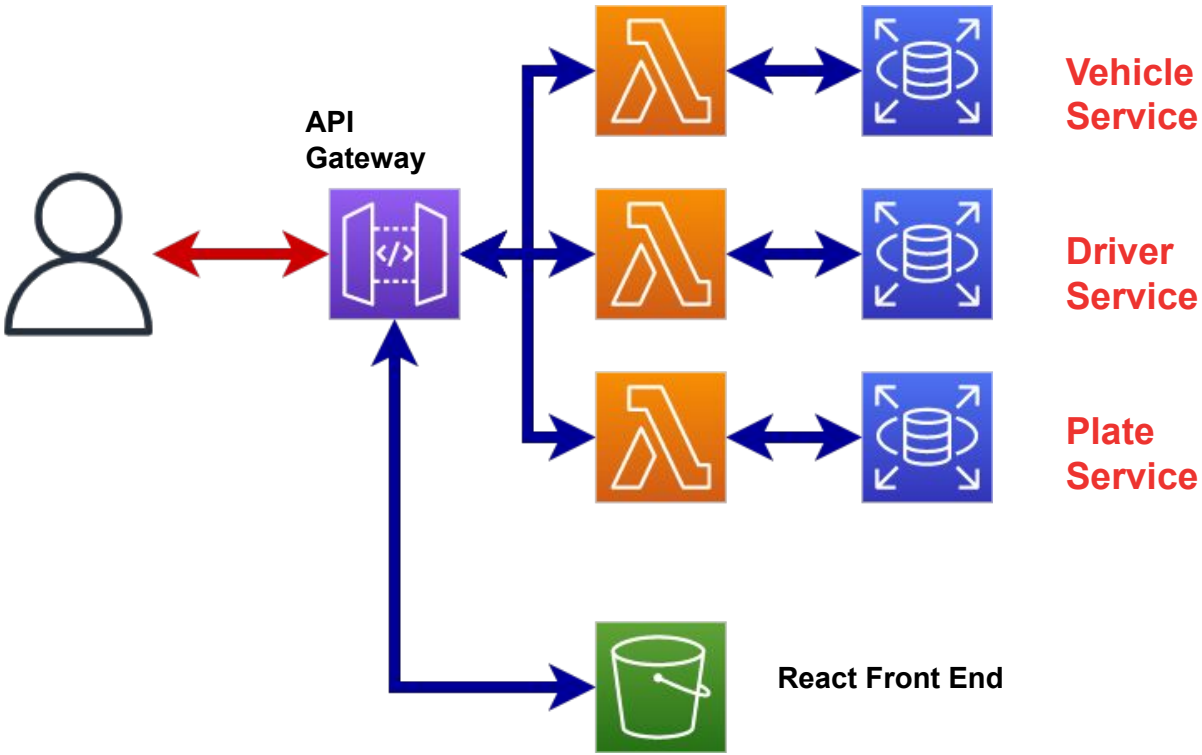


Image 11: Application Architecture Flow Chart

Application Architecture: Build

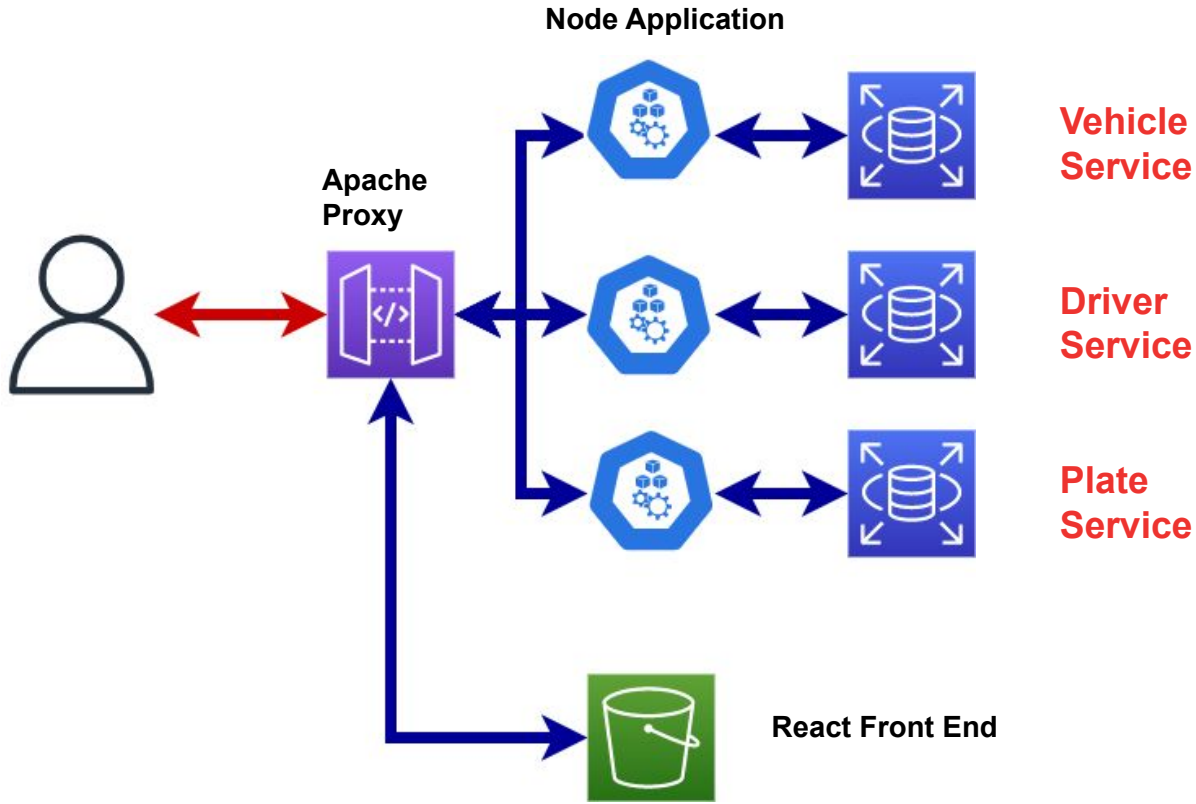
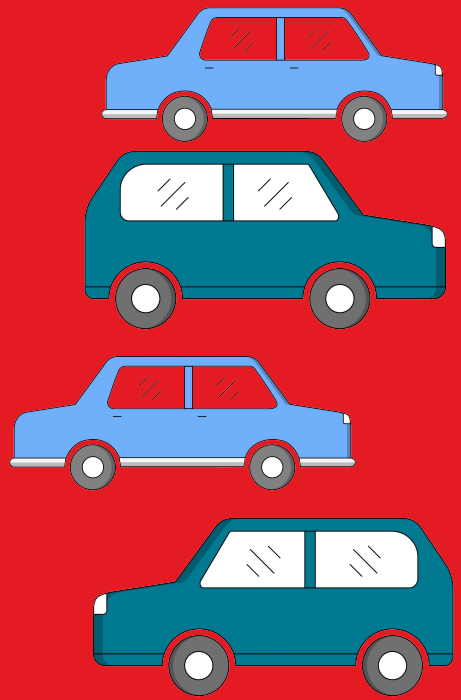


Image 12: Application Architecture Build Flow Chart

Application's User Stories

"I am a corporate vehicle fleet manager, and I want to..."

- See what vehicles a driver has
- View a Driver's information
- Add a driver
- Update a driver
- Delete a driver (and all associated plates)
- Add a plate to a driver



API Gateways

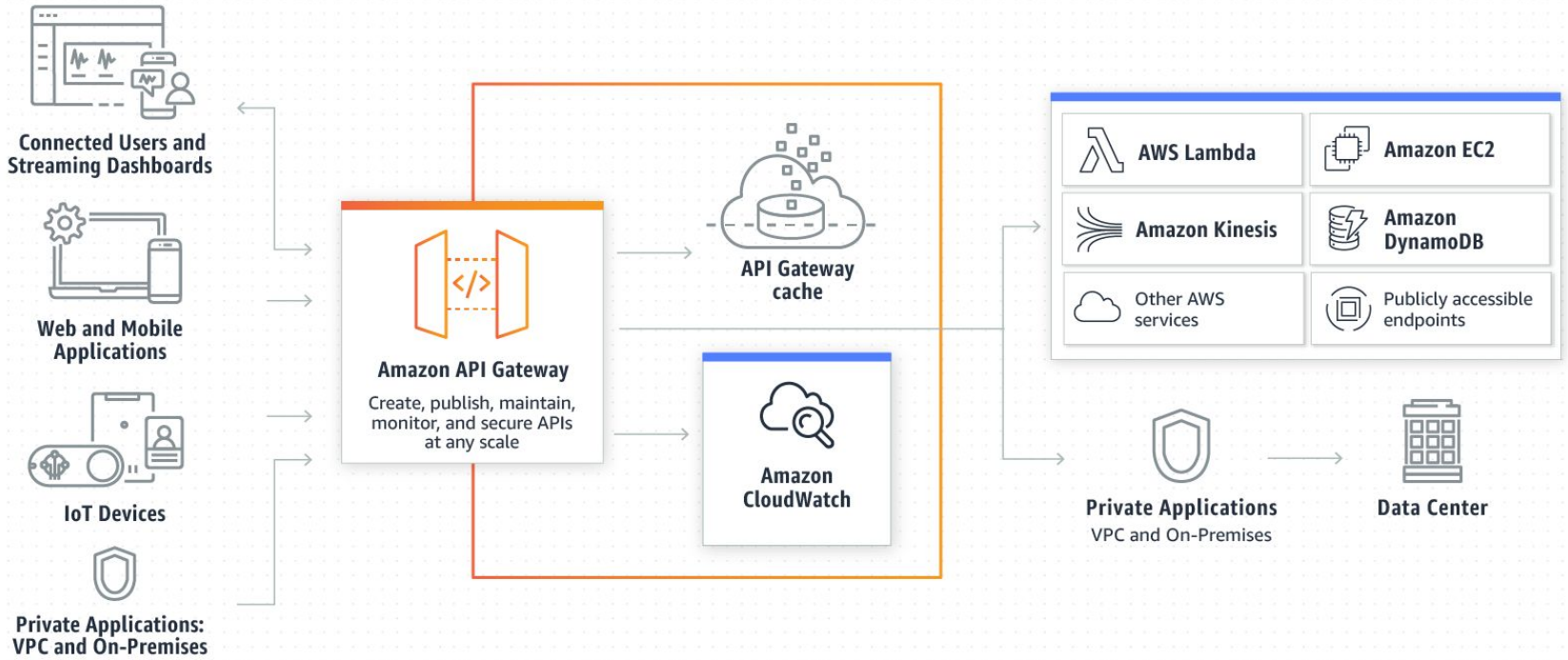
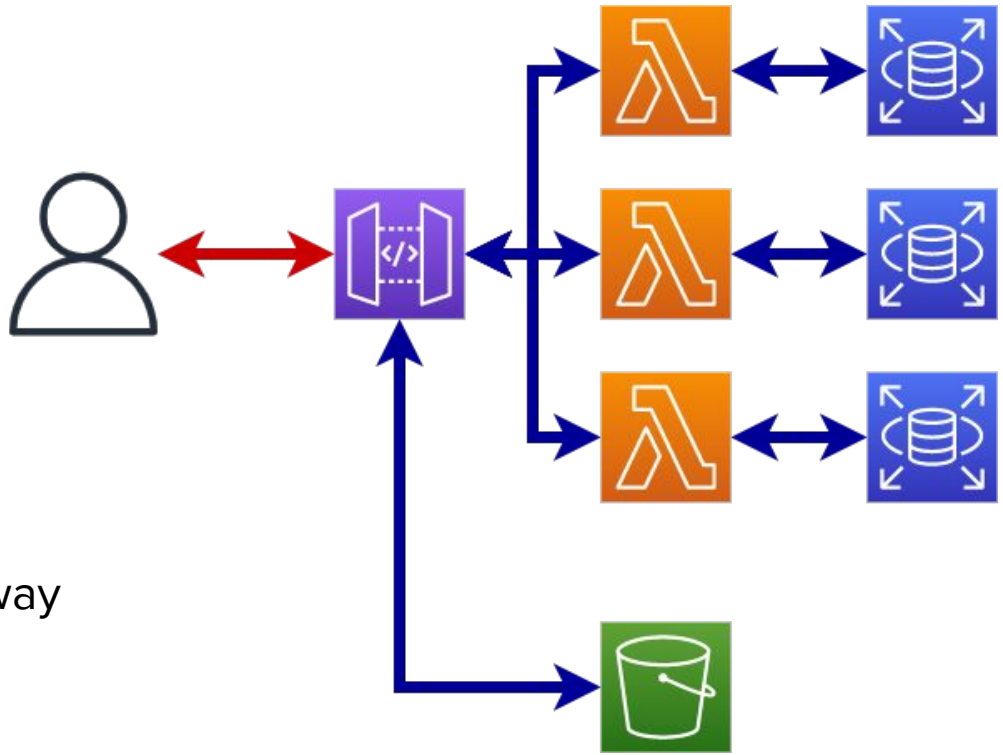


Image 13: API Gateways Flow Chart

Apache As A Gateway



We’re using apache as a stand-in for the AWS Gateway on our local machine.

Image 14: Apache As A Gateway Flow Chart

Recap

- Security
  - Access
  - Encryption
- Model Application introduction
  - Setup
  - Backend services
  - Frontend
  - High level architecture

Any Questions

Up Next

- Node & NPM
- Asynchronous considerations

Any Questions

Node & NPM

- **node.js** is a JavaScript runtime that runs on the server side.
- It is built on Chrome's V8 JavaScript engine.
- node.js comes with npm (**Node Package Manager**) which is used to manage packages and dependencies.
- npm is the largest package manager for JavaScript and contains hundreds of thousands of packages.
- With npm, developers can easily install, update, and manage packages for their projects.
- npm is a crucial tool for any Node.js developer.

Review: Asynchronous Programming

- Asynchronous programming is a programming paradigm where the program can continue to run without waiting for blocking operations to finish.
- In JavaScript, asynchronous programming is commonly implemented using callbacks, promises, and async/await.
- Async/await is a more recent addition to JavaScript and provides a way to write asynchronous code that looks and behaves like synchronous code.
- Async/await is built on top of promises and makes working with asynchronous operations easier and more intuitive.
- The async keyword is used before a function declaration to indicate that it returns a promise.
- The await keyword is used before an asynchronous operation and causes the code to pause until the operation is complete.
- Async/await provides a more concise and readable way to write asynchronous code compared to callbacks and promises.

Review: Asynchronous Programming (Continued)

```
async function fetchData() {
  try {
    const response = await fetch("https://api.example.com/data");
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error(error);
  }
}

fetchData();
```

Image 15: Asynchronous Programming Code Example

Modern Engineering

Setting Up Our Environment

Guided Walk-Through:

Setting Up Our Environment

120 Minutes

Goal:

To see how the completed app should function.

We'll be setting up our own development space tomorrow.

Instructions:

1. Git + SSH Set-up
2. Clone the repository
3. Install Apache & MySQL
4. Set up the Apache gateway
    - Virtualhost
    - Hosts file
5. Download NPM packages
6. Set up configuration files
7. Build the sample react app
8. Start each service

Guided Walk-Through:

Step 1 - Git and SSH Set-up

[Follow the instructions in this Git SSH README Guide to get started.]

Recap

- Asynchronous concerns
- Environment setup

Any Questions

Break Time

Up Next

- Designing APIs
- Experimenting in Model Application's postman collection

Any Questions

Guided Walk-Through:

Step 2 - DMV Microservices Application

[Follow the instructions in this Git README Guide to get started.]

Guided Walk-Through:

Step 1 - Git and SSH Set-up

[Follow the instructions in this Git SSH README Guide to get started.]

Day 1 Git SSH Guide

Connecting workspace to GIT using SSH

> **NOTE:** is you haven't set up your Github SSH key yet in your workspace, follow the guides provided by github for generating an SSH key, adding it to your SSH agent, and then adding that key to your GA github account. Use the following two resources,Creating an ssh key on linux and Adding your SSH key to github using the browser

We'll first need to create an SSH key on our workspaces.

Open a new terminal in your workspace and execute the following commands.

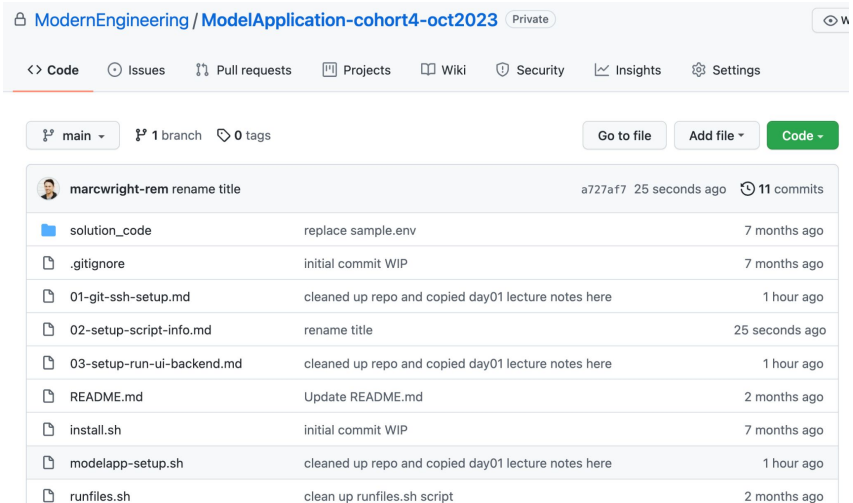
1. Let's create a new SSH key using `key-keygen` terminal tool.

> **NOTE:** be sure to change the email with the email associated with your GA github enterprise account.

Guided Walk-Through:

Step 2 - Clone the repository

1. Goto this Model Application link in your browser (*good idea to bookmark this*): http://bit.ly/3QixRF8
2. In your VM, open your Terminal and cd into your Documents folder: ***cd ~/Documents***
3. Run this command: **git clone git@git.generalassemb.ly:ModernEngineering/ModelApplication-cohort4-oct2023.git**
4. **cd ModelApplication-cohort4-oct2023/** into the app folder.
5. Open in VS Code: **code .**



Guided Walk-Through:

Steps 3 and 4 - Install and Configure Apache and MySQL

[Follow the instructions in this 02-setup-script-info.md file to set up the Model Application]

Model Application set up guide

> **NOTE:** These steps are all captured in the `modelapp-setup.sh` script. You can perform all of these steps by running the following command inside of the `D1-setup-distributed-systems` directory.

```
sudo bash ./modelapp-setup.sh
```

The `modelapp-setup.sh` script will do the following(If the script fails you can manually install using the steps below).

1. Install nodemon: a node monitoring tool we'll be using when running our backend services locally.
2. Install MariaDB: A version of MySql maintained by RedHat and easy to install on centos.
3. Run simple database setup script: We'll create a non root db user and our three databases and grant the proper permissions to those databases to our db user.
4. Install Apache: We'll install the httpd package, which is a clean install of apache through Yum.
5. Configure hosts and proxy for apache: We'll just move our preconfigured hosts and 000-default.conf files to their proper locations in /etc directory and restart apache.

Guided Walk-Through:

Step 5 - Install NPM Packages

main ModelApplication-cohort4-oct2023 / install.sh

marcicloud11 update install.sh

1 contributor

8 lines (8 sloc) | 123 Bytes

```
1  cd ./solution_code/uiSrc
2  npm install
3  cd ../servicesSrc/driver
4  npm install
5  cd ../plate
6  npm install
7  cd ../vehicle
8  npm install
```

1. We'll run the ***install.sh*** file found here. You should already have the file locally. Make sure that you're in the **ModelApplication-cohort4-oct2023** directory and run ***bash install.sh***

1. To test in the browser, go to: ***http://Localhost:80***

You should see "Service Unavailable" since we haven't started our servers yet.

Service Unavailable

The server is temporarily unable to service your request due to maintenance downtime or capacity problems. Please try again later.

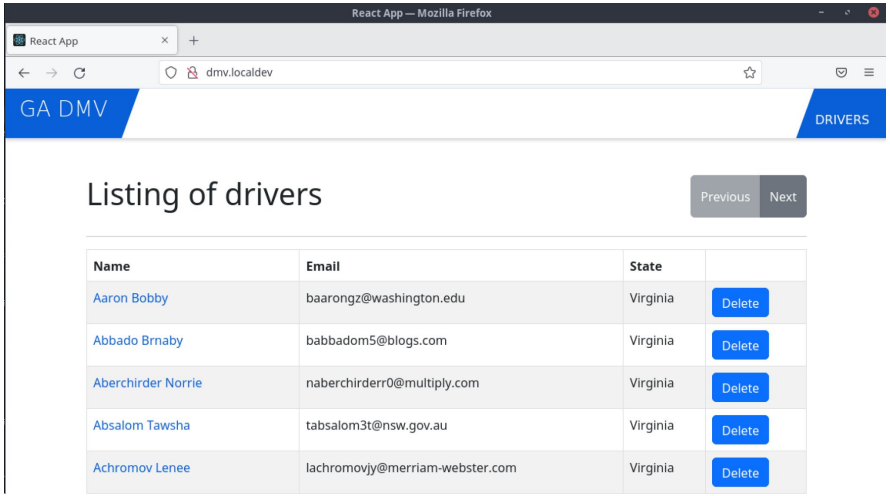
Guided Walk-Through:
Start your server engines

1. In ***solution_code/uiSrc***, rename ***sample.env*** to ***.env***

1. We'll run the **runfile.sh** file found here. You should already have this file locally. From the Model Application root directory run: ***bash runfiles.sh***

1. Use this README for reference

1. To test in the browser, go to: ***http://dmv.localdev***



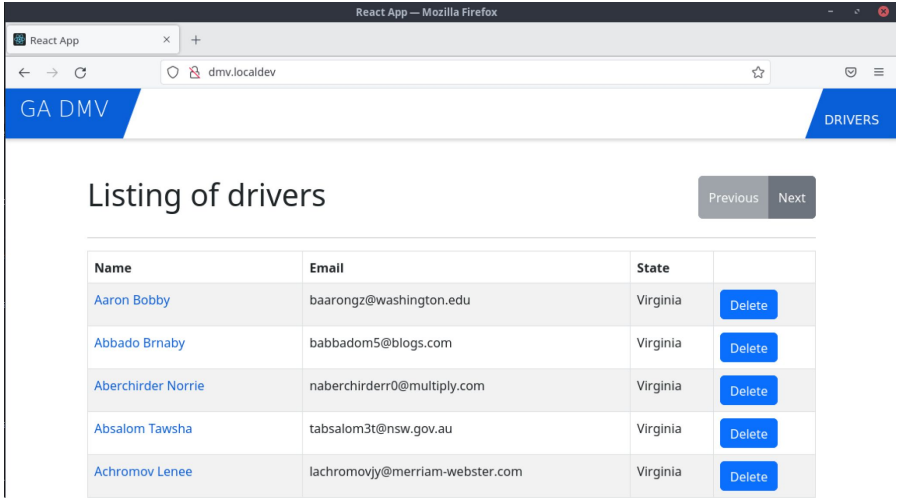
Modern Engineering

Code Exploration

Guided Walk-Through:

Explore the DMV Code

[The 03-setup-run-ui-backend.md file has detail on the app structure]



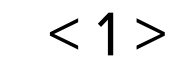
Modern Engineering

Designing our API

Considerations

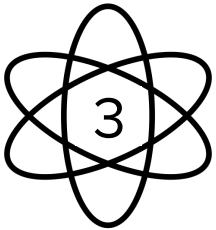
- What workflows do we need to support?
- What are the resource limits?
    - Is bandwidth limited?
    - Is back-end capacity limited?
    - Is the data complex?

API Convention Options



XML

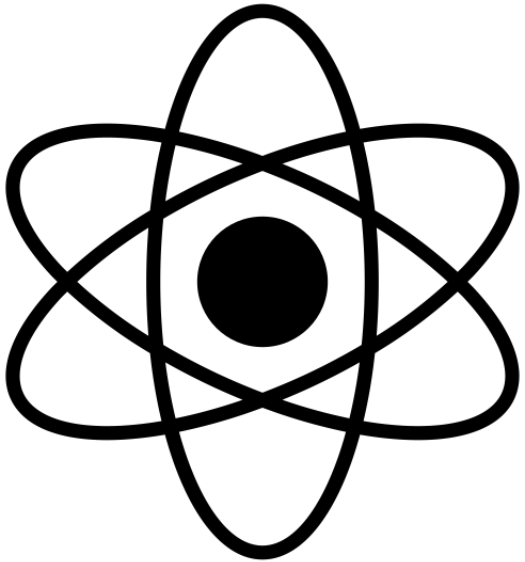
REST



GraphQL

What is GraphQL?

- **GraphQL** is an open-source query language for **APIs.**
- It's functionally an alternative to traditional **REST** approaches



REST vs. GraphQL

REST APIs

- Return all the fields from the API every time
- Require different endpoints for each service
- Require multiple requests to combine different data sources

GraphQL APIs

- Request only the fields needed
- Aggregate data from multiple sources
- Move the multi-source query logic to the server

Example of GraphQL API Interactions

POST Request

```
{
    orders {
        id
        productsList {
            product {
                name
                price
            }
            quantity
        }
        total Amount
    }
}
```

Image 16: GraphQL API Post Request Interaction

Response

```
{
    "data": {
        "orders": [{
                "id": 1,
                "productsList": [{
                        "product": {
                            "name": "orange",
                            "price": 1.5
                        },
                        "quantity": 100
                    }],
                "totalAmount": 150
            }]
    }
}
```

Image 17: GraphQL API Response Interaction

Example of REST API Interactions

POST Request

```
{
        "vin": "JH4NA21683T088489",
        "Make_Id": "8359",
        "Model_Id": 139,
        "plateText": "TYUC0RP",
        "driver_id": 86
}
```

Image 18: REST API Post Request Interaction

Response

```
{
    "Status":"SUCCESS",
    "Message": "Response successful",
    "RecordUpdated":5
}
```

Image 19: REST API Post Response Interaction


GA